EFACTLS: Effective Active TLS Fingerprinting for Large-Scale Server Deployment Characterization

Markus Sosnowski, Johannes Zirngibl, Patrick Sattler, Georg Carle, Claas Grohnfeldt, Michele Russo, and Daniele Sgandurra

***Abstract*—Active measurements allow the collection of server characteristics on a large scale that can aid in discovering hidden relations and commonalities among server deployments. Finding these relations opens up new possibilities for clustering and classifying server deployments; for example, identifying a previously unknown cybercriminal infrastructure can be valuable cyber-threat intelligence. In this work, we propose a methodology based on active measurements to acquire Transport Layer Security (TLS) metadata from servers and leverage it for fingerprinting. Our fingerprints capture characteristic behavior of the TLS stack, primarily influenced by the server's implementation, configuration, and hardware support. Using an empirical optimization strategy that maximizes information gained from every handshake to minimize measurement costs, we generated 10 general-purpose Client Hellos. They served as scanning probes to create an extensive database of TLS configurations to classify servers. We propose the Shannon Entropy to measure collected information and compare different approaches. This study fingerprinted 8 million servers from the Tranco top list and two Command and Control (C2) blocklists over 60 weeks with weekly snapshots. The resulting data formed the foundation for two long-term case studies: classification of Content Delivery Network and C2 servers. Moreover, the detection was fine-grained enough to detect C2 server families. The proposed methodology demonstrated a precision of 99% and enabled a stable identification of new servers over time. This study shows how active measurements can provide valuable security-relevant insights and improve our understanding of the Internet.**

***Index Terms*—Active scanning, TLS, fingerprinting, server classification, command and control servers.**

I. Introduction

ACTIVE fingerprinting "is the process of actively interacting with the target entity" [1] to reveal undisclosed information, like the type and version of software running on a device. Consequently, fingerprinting has many applications in the security domain, such as identifying hosts running vulnerable software, detecting network anomalies, or revealing malicious entities. One way of accomplishing this is to leverage the Transport Layer Security (TLS) protocol. TLS is currently the *de facto* standard for encrypted communication on the Internet [2]. It has evolved into a complex ecosystem due to continuous development and the need for backward compatibility [3]. Due to this, the protocol inherently provides a variety of meta-information related to client and server capabilities that they exchange during the initial TLS handshake. Previous research has leveraged these metadata using passive fingerprinting approaches [4], [5], [6]. In contrast, our work advocates active measurements, enabling engagement with any responsive server on a large scale, unrestricted by the increased encrypted communication hindering passive approaches. Moreover, it allows the creation of a comprehensive data set from a single vantage point.

Effective Active TLS Fingerprinting can help to better understand, model, and secure the Internet. If fingerprints can indicate a level of trust in infrastructure, they become valuable cyber-threat intelligence, especially given cyber criminals' increasing use of TLS [7]. Possible use cases include: (i) Intrusion Detection Systems fingerprint servers seen in network flows on-demand and compare results with known malicious fingerprints; (ii) security researchers use fingerprints from Internet-wide measurements to identify unknown threats; or (iii) regular monitoring of own servers helps to detect unintended software changes or malware infections when deviations from a fingerprint baseline occurred. Internet scanning companies like *censys.io* emphasize the need for this information as they have begun incorporating JARM [8] into their portfolio (according to their data definitions [9]). JARM is an additional example of an open-source TLS server fingerprinting tool that utilizes data similar to the one presented in this work and has recently gained prominence in its usage.

This work demonstrates that only effective fingerprinting can supply the necessary information to be valuable for the mentioned use cases. It is an extension of the study described in [10], which provides a long-term analysis showcasing the applicability and performance of detection use cases supported by active TLS fingerprinting, along with an assessment of the effectiveness of their data collection.

In this work, we investigate (i) how to construct a similarity relation among TLS server deployments, (ii) how effective scanning configurations can be found while minimizing the measurement costs, and (iii) how active TLS fingerprinting applications perform on a large scale. To this end, we introduce

Manuscript received 9 October 2023; revised 29 January 2024; accepted 1 February 2024. Date of publication 9 February 2024; date of current version 12 July 2024. The associate editor coordinating the review of this article and approving it for publication was I. Drago. *(Corresponding author: Markus Sosnowski.)*

Markus Sosnowski, Johannes Zirngibl, Patrick Sattler, and Georg Carle are with the Department of Computer Science, TUM School of Computation, Information and Technology, Technical University of Munich, 80333 Munich, Germany (e-mail: sosnowski@net.in.tum.de).

Claas Grohnfeldt, Michele Russo, and Daniele Sgandurra are with Huawei Munich Research Center, 80992 Munich, Germany.

Digital Object Identifier 10.1109/TNSM.2024.3364526

EFACTLS, offering an effective method for active TLS fingerprinting.

The paper presents the following contributions:

(i) a reasoned selection of TLS handshake features for fingerprinting the TLS stack and their encoding in an extendable and shareable format;
(ii) a methodology for finding or tailoring effective TLS Client Hellos (CHs) for fingerprinting use cases and 10 pre-made general-purpose CHs that maximize complementary information extraction from servers;
(iii) the use of the Shannon Entropy as metric to measure the information obtained from active Internet measurements;
(iv) a classification approach that utilizes active TLS fingerprinting to predict server classes;
(v) a one-year-long measurement study to validate the methodology, demonstrate improvements to the related work JARM, and showcase the potential of active TLS fingerprinting through two case studies: Content Delivery Network (CDN) and Command and Control (C2) server detection;
(vi) an extended analysis that reveals active TLS fingerprinting is fine-grained enough to predict concrete C2 server families; and
(vii) data of the experiments and the open-source scanner released publicly, to enable reproducible results and to support the community.[1]

This work is an extension of [10], published in the Network Traffic Measurement and Analysis Conference (TMA) in 2022. Contributions (iii), (iv), and (vi) represent novel additions compared to the previous version. Additionally, we conducted the longitudinal study on a significantly larger and more recent time frame to demonstrate the applicability of our analyses on new data. In this work, we provide fresh insights into related research and enhance the existing content by incorporating the Entropy metric, comparing additional related works, and adapting findings to the new data set. Compared to our previous work, we switched to the Tranco [11] top list as input for our measurement study due to the discontinuation [12] of the Alexa top list service.

II. Related Work

The large amount of metadata from TLS handshakes has been used in multiple passive traffic classification and fingerprinting related works [4], [5], [6]. In the context of the Transmission Control Protocol (TCP), fingerprinting with active scans has been successfully used by [13], [14] and [15] to detect the Operating System (OS) on a remote server. Recently, Zirngibl et al. [16] applied active fingerprinting on the QUIC [17] protocol to fingerprint and analyze libraries. Like our CH selection, Greenwald and Thomas [13] used the Entropy from the information theory as a metric to minimize the number of probes needed for classification.

A. Active TLS Fingerprinting Approaches

This section presents related TLS scanning and fingerprinting approaches. We selected them because they all extract metadata from the TLS layer that can be used for fingerprinting. Additionally, they have in common to implement specialized probing mechanisms that are able to extract more information from the TLS layer than it is possible by conducting a single or trivial handshakes. We summarize our findings on related approaches and tools in Table I. None of the other active TLS fingerprinting works has used Entropy or an equivalent metric to evaluate or optimize their approach.

TABLE I
Summary of Related TLS Scanning and Fingerprinting Tools

SSLyze	testssl.sh	DissecTLS	TLS Prober	JARM	EFACTLS	
✗	✗	✗	○	✓	✓	Fixed Probes / Scan Duration
✓	✓	✓	✗	✓	✓	Covers TLS Configuration
✗	✗	✓	✓	✓	✓	Covers TLS Interpretation
✓	✓	✓	✗	○	✓	Supports TLS 1.3
430[2]	155[2]	24[2]	10–295[1]	10	10	CH Usage
[18]	[19]	[20]	[21]	[8]		Reference

Legend: ✓ Yes ✗ No ○ Partly
[1] TLS Prober has a "quick scan" mode that stops scanning as soon as 10 probes suggest the same implementation.
[2] The average number of sent CHs observed in a study on the Tranco top 10k domains published in Ref. [20].

To the best of our knowledge, the closest related work we can directly compare ourselves to is the JARM tool developed by Althouse et al. [8]. It is a popular open-source tool for TLS server fingerprinting. Compared to this work, our tool differs in the concrete choice of CHs and the extracted features from the TLS handshake. They use 10 custom-defined CHs for fingerprinting that "have been specially crafted to pull out unique responses in TLS servers" [8]. In contrast to this work, they do not complete the TLS handshake, only use unencrypted data, and do not consider TLS alerts nor extension data besides the Application Layer Protocol Negotiation (ALPN) protocol; hence, they use only part of the data offered by TLS 1.3. We will show in Section V-F that JARM also enables C2 server detection; however, the data suggested by this paper can improve the approach's effectiveness.

A fundamentally different approach for collecting TLS data is to dynamically search for each piece and change the scanning probes during the fingerprinting based on already learned information. A scanner can adapt scanning probes on the fly if it contains a model interpreting the TLS stack on a server. The scanner would use the model to interpret previous server responses and generate a new probe likely to fill the gaps in the already collected data. However, the quality of the collected data relies on the quality of the underlying model because it has to explain any observed behavior. Adapting the scan per server can result in a variable and unpredictable scan duration. In contrast, the fixed probing from this work has a constant scan duration independent of the inner workings of TLS server implementations. The only requirement is that servers behave consistently. The related work DissecTLS [20] shows that it is possible to implement such a dynamic scanning approach efficiently enough to use it for large-scale measurements.

[1] https://tumi8.github.io/active-tls-fingerprinting/

DissecTLS provided a higher C2 detection precision compared to this work at the cost of sending more requests (24 ch per server on average [20]). Similarly, TLS server debugging tools like testssl.sh [19] or SSLyze [18] dynamically collect data about the TLS servers. Results presented in DissecTLS [20] demonstrate that the data from both tools can be used for active TLS fingerprinting, although it is necessary to sanitize their output first. However, they used many requests per server, making Internet-wide scans time-consuming and ethically questionable. Additionally, their focus on the configurable part of TLS on servers (e.g., supported cipher suites) results in neglecting fingerprintable implementation-specific features like the extension order. In contrast, TLS Prober [21] solely focuses on such features. Mapping out edge cases of the TLS protocol enabled the author to fingerprint only the TLS library. The tool uses up to 295 probes to reveal implementation-specific behavior, such as the mapping of TLS messages to TLS records or how the implementation reacts to erroneous or unusual input. Section V-D will present how additional data sources can improve CNC detection. We will present how additional data sources can improve the C2 detection in Section V-D. It is possible that the data obtained with TLS Prober can be used in conjunction with ours to improve detection further in the future. However, the tool has not been updated recently and only supports TLS 1.2.

B. Related Observations on the TLS Ecosystem

Chung et al. [22] investigated the usage of the Online Certificate Status Protocol (OCSP) stapling from different Web servers. He observed Nginx servers, which did not return OCSP responses to the first client connecting, appending the information only to consecutive requests. These implementations did not pre-fetch the information nor wait until the Certificate Authority (CA) returned the necessary OCSP response they could forward to the client. Hence, from the client's point of view, the presence of stapled OCSP responses is non-deterministic. Their observations on OCSP stapling align with our observations of the non-deterministic presence of the Status Request extension because it is currently only used by servers to announce stapled OCSP responses (cf., [23]).

Gigis et al. [24] investigated Hypergiants, including CDNs, over seven years. They showed the increasing role of servers deployed in Autonomous Systems (ASs) not managed by the CDN to influence and localize CDN traffic to the user. Their results align with ours because we could also find server deployments outside the networks managed by the CDN, and they found indicators of reverse proxies that influence the measurement results. Their results rely on servers correctly offering identification material in the certificates. On the other hand, this work is more subtle and can identify deployments where the information is deliberately hidden (e.g., to detect C2 servers).

Papadogiannaki and Ioannidis [25] have used JARM to study the evolution of C2 servers based on their TLS fingerprints. Like us, they used the Feodo Tracker as input but compared the C2 fingerprints only with ones obtained twice from the top 10k Majestic domains instead of weekly top 1M scans. Similar to this work, they concluded that it is possible to identify C2 servers based on their TLS fingerprint; however, they noticed that fingerprints overlapping with legitimate servers increased over time. They found that it is vital to update a fingerprint database often or to use a more effective fingerprinting approach; otherwise, the detection becomes less effective. We also conclude the necessity of an up-to-date database and an effective approach.

III. Methodology

The TLS protocol family "is the backbone of secure communication over the Internet" (as introduced in more detail by Holz et al. [26]) and is currently the *de facto* standard for encrypted communication [2]. This work exploits the TLS protocol to discover similarity relations among servers by fingerprinting their Server Behavior. We define *Server Behavior* as the totality of the capabilities, the interpretation (deviations from the standard or implementation of undefined parts, such as the order of extensions) and the configuration of the TLS on a server, which can influence the outcome of the TLS handshake. Our work assumes that every TLS server has a specific Server Behavior that depends on the implementation, capabilities, and configuration of the TLS library, hardware, and application utilizing the TLS. Identifying these behaviors allows characterizing server deployments either directly or in conjunction with additional data (e.g., obtained on the Hypertext Transfer Protocol (HTTP) layer).

Clients initiate TLS handshakes, and servers only need to react to the initial handshake request (e.g., a server chooses one cipher from a list that the client previously proposed). Therefore, the Server Behavior we want to fingerprint is not directly revealed by the server; only the reaction to different requests (i.e., CHs) is visible. Using multiple CHs increases the acquirable knowledge and coverage of the Server Behavior. For each CH, we collect the TLS version, cipher, and TLS extension data from different types of TLS messages to construct the fingerprint. We only initiate handshakes with TLS versions 1.0 to 1.3 but construct a fingerprint from any version the server sends as a response.

This work proposes a methodology for capturing a part of the Server Behavior by sending a fixed number of specifically crafted CHs to a server, extracting features as string-encoded information for each CH, as detailed in Section III-A, combining these features to a fingerprint according to Section III-B, and use fingerprints for a threshold-based classification according to Section III-C. Section III-D describes Entropy as a metric to measure the amount of information collected via active scanning and fingerprinting.

A. Features Extracted From TLS Handshakes

Given a CH, we extract features from a single handshake in a textually encoded format for fingerprinting.

Features are the selected version, cipher suite, received alerts and extension data. We extract extensions as an ordered list of key-value pairs from the *Server Hello*, *Encrypted Extensions*, *Certificate Request*, *Hello Retry Request*, and

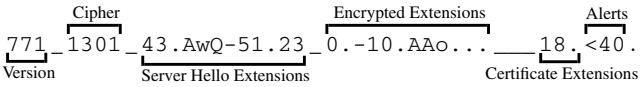
TABLE II
TLS EXTENSIONS WHERE ACTIVE TLS STACK FINGERPRINTING USES THE DATA CONTAINED IN THE EXTENSIONS FOR FINGERPRINTING

ID	Name	ID	Name
1	Max Fragment Length	19	Client Certificate Type
7	Client Authentication	20	Server Certificate Type
8	Server Authentication	24	Token Binding
9	Cert Type	27	Compress Certificate
10	Supported Groups	28	Record Size Limit
11	EC Point Formats	43	Supported Versions
13	Signature Algorithms	47	Certificate Authorities
15	Heartbeat	50	Signature Algorithms Cert
16	ALPN	51	Key Share (only selected group)

TLS extension IDs as specified by IANA [27].

Certificate TLS messages. The extension identifier served as the key and the content of the extension as the value. However, we only included the content of several hand-picked extensions (Table II) and left the value empty otherwise.

An example of a textually represented fingerprint is



In the subsequent paragraphs, we will discuss the reasons for including each feature. The version, selected cipher suite, and used extensions directly depend on the capabilities and the configuration of the TLS library used by the server and are, therefore, part of the fingerprint. The content of each extension can contain information independent of the Server Behavior, and its fingerprinting value depends on the respective extension. We excluded information depending on the current TLS session, a specific server instance, or the TLS certificate. We inferred the content of the extensions listed in Table II as relevant features for fingerprinting based on an analysis of the respective specifications [23], [28] and our observations in our scans. An exception to this schema is the Key Share extension, where we removed the session-specific part and only kept the selected group used for the Diffie–Hellman key exchange. The fingerprints are defined in a format that can easily be adapted (e.g., to include values specified in future extensions). We included the TLS alerts sent by the server in the fingerprints because error handling is implementation-specific. However, we would not create a fingerprint if the TCP layer caused the error. The current approach cannot differentiate whether the error was part of the Server Behavior or a nondeterministic failure of the TCP stack. We considered the order of extensions valuable and implementation-specific information that we included in our fingerprints. The Status Request extension was nondeterministic in our measurements (Section V-B); therefore, we removed the extension from the fingerprints, trading the information about the OCSP stapling support of a server for consistency.

B. Fingerprinting With Multiple Requests

During the development of our methodology, we noticed that more than a single response from a server was needed to provide good results in our experiments. Therefore, this work combines data from multiple server responses to construct the TLS fingerprint of the Server Behavior.

While a single CH reveals only a potentially small request-dependent subset of the information about the target server, multiple request–response pairs allow the collection of complementary information and, thus, a more complete picture of the Server Behavior. Increasing the number of CHs is a trade–off between learned information and measurement costs. However, the benefit of sending multiple CHs decreases with every additional CHs one sends because of the limit to which a Server Behavior can influence the TLS handshake. Moreover, the number of CHs should be limited based on time, resources, and ethical factors. Hence, the input set *CH* of CHs is an optimizable parameter influencing the effectiveness of fingerprinting. Let $f(s, c)$ return the features from a server s given a specific CH c in the format described in Section III-A; then, the server fingerprint is defined as

$$fp(s) = \bigcup_{c \in CH} (c, fp(s, c)).$$

Server responses are a reaction to the initiating handshake request; e.g., a server can only choose a cipher suite that the client previously proposed. Consequently, the features obtained with a single CH are only comparable in the context of the same CH; hence, the CH used to generate each part of a fingerprint is a feature of said fingerprint. We never compared information obtained with different CHs because we need to know what combination of parameters in the CH has caused a particular response. In our implementation, we string concatenated the features obtained with each CHs in a consistent order and stored the CHs separately.

In conclusion, the number of requests and the design of different CHs are crucial parameters that can be optimized to maximize the amount of collectible information while minimizing measurement costs and respecting ethical aspects. We experimentally define the CHs of this work in Section IV.

C. Threshold-Based Classification

This work uses a threshold-based classifier to implement different active TLS fingerprinting use cases. Well-known classification metrics allow the evaluation of this classifier, particularly precision and recall.

The classifier works as follows. First, all fingerprinted servers are labeled according to the use case and split into training and evaluation sets. The labels can be unspecific (e.g., whether a server is considered a C2 server or not) or more precise (e.g., a TrickBot C2 server). Then, we calculate a prediction for each fingerprint using the training set. The prediction is the probability that a particular fingerprint predicts a label. We estimated each probability by dividing the number of times a fingerprint was seen with each label by the total number of occurrences of the respective fingerprint. Afterward, we applied the predictions to the evaluation set to classify each server. However, only predictions with a probability above the configurable threshold were considered. For example, a particular fingerprint appeared 100 times in the training set: 12 times from servers labeled TrickBot and 30 times QakBot. Observing the said fingerprint in the evaluation

set would result in a probability of 12% TrickBot and 30% QakBot. For a threshold of 20%, the classifier would predict the QakBot class; for a threshold of 50%, the classifier would produce no prediction. Lastly, the metrics of precision and recall are computed based on the evaluation set. Both metrics are defined only in the context of a specific label. Whenever the classification matches the label, it is counted as True Positive (TP); if they do not match, as False Positive (FP). The True Observations (PPs) are the total number of times a label occurred. The precision and the recall are defined as $\frac{TP}{TP+FP}$ and $\frac{TP}{PP}$, respectively.

Intuitively, precision is the rate of correct classifications of a label. Our classifier assigns a class to not every server in the evaluation, expressed in a lower recall. The recall gives the rate of correctly identified servers compared to the total amount of servers that could have been identified. The threshold serves as a tuning parameter that can increase the precision at the cost of a sometimes lower recall.

In summary, the threshold-based classifier is a simple but effective approach to implement and evaluate the fingerprinting use cases shown later in Section V.

D. Entropy as Metric for the Collected Information of Active Measurements

In 1948, Shannon [29] defined a theoretical approach for measuring the amount of "information" contained in arbitrary communication. We propose to apply his definition to model the amount of information collected via active measurements. Entropy is a metric that assigns an observation a low value if it is likely to be observed; on the other hand, an observation that rarely occurs has a high value. Intuitively, this also reflects our understanding of fingerprinting: a fingerprint that is the same for every server has no value. However, a fingerprint observed only from a few servers is very valuable because it indicates a commonality. A downside of Entropy is that it cannot reflect whether the collected information is useful. For example, an approach producing a unique fingerprint for each server would result in the maximum possible Entropy. However, such fingerprinting would have no value because it does not reveal relations among the servers. Nevertheless, Entropy can be an effective metric to evaluate actively collected data if the usefulness is shown through other means; e.g., a measurement study like the one we perform in Section V.

C. Shannon abstracted a data source as a Random Variable $\mathcal{Q}$ that emits symbols $x \in \mathcal{X}$ with a certain probability $p(x) \in [0,1]$. He interpreted the "uncertainty" of each symbol as its information content $I(x) = -log_2\, p(x)$. The overall information, defined as Entropy, of a data source $\mathcal{Q}$ is the average across all possible symbols:

$$H(\mathcal{Q}) = \sum_{x \in \mathcal{X}} p(x) I(x).$$

The Entropy definition can be applied to active measurements by modeling the set of scanned servers as a single data source and Random Variable $\mathcal{Y}$. Every collected datum—in our case, a TLS fingerprint $f_p \in FP$—is abstracted as a symbol that $\mathcal{Y}$ emits. The probability $p(f_p) \in [0,1]$ that a specific datum occurs is unknown. However, we can measure the rate at which it occurred in a single measurement by dividing the number of observations from a specific datum $o(f_p) \in \mathbb{N}$ by the total number of scanned servers $t \in \mathbb{N}$. According to the law of large numbers [30], the occurrence rate approximates the actual probability, given that the set of scanned servers is large enough. Hence, we can measure the collected information from a single active measurement as

$$H'(\mathcal{Y}) = -\sum_{f_p \in \mathcal{FP}} p'(f_p) \cdot log_2\, p'(f_p)\,, \text{with } p'(f_p) = \frac{o(f_p)}{t}.$$

To conclude, the Shannon Entropy can be used to measure the collected information of an active measurement. Moreover, Entropy is a metric that allows to compare the effectiveness of different scanning and fingerprinting approaches. We will discuss the limitations of the Entropy metric in Section VI-G.

E. Active Measurements Under Ethical Considerations

We have used an active measurement pipeline based on established tools and by following basic ethical principles.

The pipeline takes a list of IP addresses, domains, and CHs as input. MassDNS [31] and a local Unbound [32] server resolve the domains to their IPv4 and IPv6 addresses, resulting in a set of (IP address, domain) pairs we call targets. We fingerprinted with multiple CHs; thus, the final scan input was a randomly ordered cross-product of the target list and the ten CHs, resulting in (IP address, domain, CH) triples. IP addresses can be augmented with a TCP port that should be used instead of the default 443 port. We used the TUM goscanner [33] to perform a TLS handshake for each triple and collect the required fingerprinting data. The TUM goscanner is a TLS scanner designed for Internet-wide usage and initially implemented by Amann et al. [34]. If a domain name was available for an IP address, we used it as the Server Name Indication (SNI). We designed a custom TLS library based on the Golang standard library that allows the definition of arbitrary CHs as input for each TLS connection and extracts sought TLS handshake metadata. Both the scanner and library are open-sourced [33]. We additionally scanned each target with the DissecTLS [20] approach to compare their effectiveness.

We reduced the impact on third parties by following the best practices described by Durumeric et al. [35]. Our work does not harm individuals or reveal private data, as [36] and [37] cover, and focuses on publicly reachable services. We used rate limiting, maintained a blocklist, used dedicated scan servers with abusive contacts, configured informative rDNS entries, and hosted websites that informed scanned parties about our research. We also provided contact information for further details or scan exclusion. Additionally, because we scan the same target with multiple requests, we limit the interference by spreading the requests over an extensive time frame (i.e., two days in the longitudinal study).

IV. Systematic Design of Client Hellos

The internal mechanism of TLS servers is a black box for active scanners. Without knowledge about the implementation

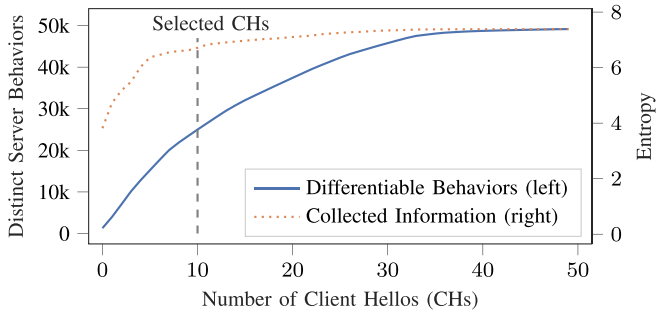


Fig. 1. Experimentally determining the trade-off to increase the number of CHs versus the ability to distinguish Server Behaviors (measured in distinct fingerprints). We selected a set of 10 CHs for our following analyses.

of every TLS server, finding the best method for fingerprinting is impossible. However, more effective fingerprints can be developed by optimizing their distinctiveness. We propose herein an empiric design of CHs by analyzing a large pool of randomly generated candidates to find an optimal subset maximizing a given metric. We choose the number of distinguishable servers as a metric to find general-purpose CHs usable for a wide range of use cases. Alternatively, we could have used Entropy; however, at the point of conducting this experiment (published in [10]) we had yet to introduce the Shannon Entropy (Section III-D) as a metric. If the use case is known (e.g., detecting CDN or C2 servers), a different strategy could be minimizing the necessary probes needed for a classification. We will later revisit the use-case-driven design regarding C2 servers in Section VI. We have open-sourced the general-purpose CHs and their generation code as part of our scanner [38].

Our systematic design of CHs worked as follows:

(i) we randomly generated 5000 CHs each from the feature space our TLS scanner supported and the complete feature space as defined by IANA [39];
(ii) we fingerprinted top list servers[2] iterating over the 10000 CHs with a maximum of 13 CHs per server to gain a first impression of good-performing CHs; and
(iii) selecting the best-performing CHs from the previous measurement, we conducted a second measurement of the top list servers and fingerprinted each target with 50 CHs.

We choose the prime number 13 and a round-robin algorithm to increase the variation of the different sets of CHs sent to a single server. Scanning with 50 CHs per server was a pure trade-off between scanning speed and data quality (the scan took more than four days).

Fig. 1 shows the number of Server Behaviors fingerprinting with a subset of the 50 CHs could distinguish and the collected information according to the Entropy (introduced in Section III-D). A simple hill-climbing algorithm generated the subsets, maximizing the number of distinguishable Server Behaviors. The algorithm worked as follows: it iteratively selected the next CHs that most increased the number of distinct Server Behaviors. We considered only servers for which every CH produced a fingerprint to remove the potential bias from nondeterministic TCP errors. Fig. 1 shows that every added CH enabled the differentiation of additional Server Behaviors; however, the information gain decreased the more CHs used. We could not reach the upper limit of distinguishable behaviors and collected information. Based on this analysis, we selected 10 general-purpose CHs with a good performance distinguishing Server Behaviors. We thought that selecting 10 CHs was adequate for our use cases because the Entropy increase was low when using more than 10 CHs (according to Fig. 1), we can directly compare related work, and the number seemed acceptable for Internet scanners like `censys.io` already fingerprinting with JARM [9]. However, Section VI-C will illustrate that the number can be lower for specific use cases and still provide good results. We only manually adapted some cryptographic parameters of these CHs that were too CPU-expensive, such as the 512-bit version (`secp521r1` [40]) of the elliptic curve domain parameters for the precomputed TLS 1.3 Key Share. Enabling `secp521r1` would have more than doubled our scanning time.

Through the experiment described in this section, we gained a set of 10 general-purpose CHs that we will use for fingerprinting servers on the Internet in the following Section V. They are a good trade-off between limiting the number of requests and the resulting impact on the scanned infrastructure and providing a high distinctiveness of the Server Behaviors.

V. Longitudinal Study of Toplists and Blocklists

To investigate the applicability of TLS fingerprinting on the Internet, we measured top lists and two C2 blocklists over one year. The following sections analyze the stability of the fingerprints, apply the methodology to detect CDN and C2 servers, and compare it to related work. The two case studies were selected to have one with a significant sample size where the ground truth can be verified and one where the value of the study is high but the sample size is small. We published the raw measurement data that served as the basis of this section under [41].

A. Data

We scanned servers from a top list and two blocklists over 60 weeks using 57 weekly snapshots starting July 4, 2022. Three scans failed due to infrastructure problems and an issue in the scanning scripts. We skipped these weeks.

Table III presents the number of scanned servers. A *target* is the scanned combination of IP addresses, TCP port, and domain name. We used the Tranco [11] top 1M list as the top list. The last 30 days were used for the SSLBL [42], while the current list was utilized for the Feodo Tracker [43]. We took a considerable time frame for the SSLBL because the published ports and IP addresses are just indicators and the actual blocklist consists of certificate hashes. In the following analyses, we only considered servers to be blocked by the SSLBL if they returned one of the blocked certificates. The combined list of around 1.8M weekly targets was taken as input to the scanning pipeline, as described in Section III-E. The scanning probes are the 10 CHs designed in Section IV

[2] At the time of the experiment, published in [10], we were using the Alexa and Majestic instead of the Tranco [11] top list.

TABLE III
TOTAL NUMBER OF COLLECTED DATA SAMPLES OVER 60 WEEKS FOR THE LONGITUDINAL ANALYSES. ALSO SHOWS THE NUMBER OF DISTINCT TARGETS AND DOMAIN NAMES THE DATA COVERS ($M = 10^6$)

Source	Scanned			Successful		
	Total	Targets	Domains	Total	Targets	Domains
Tranco	102.2M	8.4M	1.9M	78.1M	6.2M	1.5M
Feodo	155.8k	4.5k		13.4k	1.3k	
SSLBL	6.0k	628.0		323.0	93.0	
Total	102.4M	8.4M	1.9M	78.1M	6.2M	1.5M

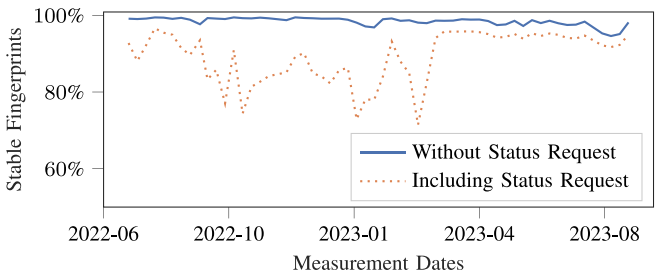


Fig. 2. Percentage of targets with the same TLS fingerprint on the $n-1$ and nth measurement in relation to the total targets fingerprinted on both weeks. The Status Request extensions are responsible for the most unstable fingerprints, with drops under 90% caused mainly by Cloudflare.

and the 10 CHs modeled after *JARM* [8], which allows us to compare both approaches. However, unless stated otherwise, the following analyses are only based on the 10 CHs designed for this study. We consider targets successfully fingerprinted if a fingerprint for each CH was collected. The total number of targets was less than the sum of each list due to a small overlap on the blocklists.

B. Consistency of TLS Fingerprints

The fingerprints only provide value for identification purposes if they can be unambiguously assigned to a server, and this assignment does not change; in other words, it is stable.

For each measurement, many servers were already seen in the last measurement ($\approx 66\%$ each week); hence, their fingerprints can be compared over time. Fig. 2 shows the relative number of targets remaining stable during each measurement. On average, the targets remained stable 99% of the time. The stability drops to an average of 89% if the fingerprinting includes the Status Request extension. In these cases, the presence of the extension is nondeterministic. Interestingly, after March 13, 2023, the stability improved even when including the Status Request extension. We did not change our methodology; the stability improvement was due to a change seen from Cloudflare servers. Before the date, we observed inconsistent results from Cloudflare of up to 14%, and after the date, around 3%. Cloudflare is very dominant in our data and was able to cause such statistically significant effects (i.e., the Cloudflare AS was able to serve 29% of the evaluated Tranco domains at least once). We confirmed the behavior change with Cloudflare; apparently, they improved their "OCSP Fetcher service" [44] during this time, increasing the number of certificates with OCSP staples available.

TABLE IV
SERVERS SEEN WITH A TLS FINGERPRINT FROM THE RESPECTIVE CDN. A STRONG CORRELATION BETWEEN BOTH CAN BE SEEN

	Akamai	Alibaba	Cloudflare	Fastly
Targets with a TLS fingerprint attributable to a CDN				
Total	105 638	84	1 980 608	74 007
IP addresses	36 972	59	215 374	10 533
Subset observed from ASs owned by the CDN				
Total	105 222	81	1 978 913	60 769
IP addresses	36 734	56	214 728	1 600
Subset observed from different ASs, but with access to CDN content				
Total	118		1 435	
IP addresses	86		469	
ASs	16		32	

This analysis concludes that Status Request extensions should not be considered for obtaining useful fingerprints. However, TLS fingerprints are, without the extension, a very stable and consistent feature to identify servers. The subsequent sections analyze how fingerprinting can reveal whole deployments of similar servers.

C. Case Study: Detecting CDN Server Deployments

A core assumption about active TLS fingerprinting is that it reveals groups of similar server deployments. We tested this assumption by analyzing the fingerprints of four major CDNs. On the one hand, these are TLS-enabled servers deployed by a single actor on a large scale. On the other hand, we can verify if a server is a part of the CDN, effectively producing a ground truth. Moreover, we found servers outside of the ASs operated by the CDN that served CDN content.

The analyzed CDNs have in common that they use their own ASs to deploy servers and CDN caches. Hence, we could identify them by their AS, which we determined through Border Gateway Protocol (BGP) dumps downloaded from Routeviews [45] and Pyasn [46]. The content served by CDNs is independent of the actual server or the IP address. The CDNs decide on other criteria, like the SNI, which content they should return. Similarly, the CDN selects the proper TLS certificate for the requested domain. Hence, we can evaluate whether or not a server is a valid CDN cache with our TLS scanner. The server is verified as CDN cache if it completed a valid TLS handshake for a domain we have manually observed to be cached by the CDN and, therefore, proves possession of a valid certificate and the respective private key.

This analysis focused on the CDNs of Cloudflare, Fastly, Akamai, and Alibaba, as they provided promising results in our previous work [10]. The CDN fingerprints were collected based on the scanned IP addresses located within their respective AS. The HTTP Server header enhanced the mapping, as described by Gigis et al. [24]. Note that the AS was sufficient for Fastly to detect their CDN servers. Table IV lists an overview of the results showing a strong correlation between the TLS fingerprint and the CDNs. Almost all targets with a CDN fingerprint were located within a CDN AS. In addition, several targets we falsely assigned to one of the

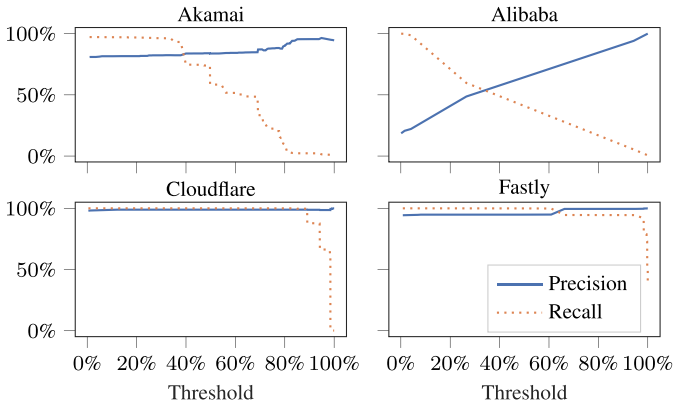


Fig. 3. Overview of the performance of the CDN server classifier for different thresholds. Training and evaluation sets were the combined data over each measurement. Based on this analysis, we selected a 40% threshold for the detection over time.

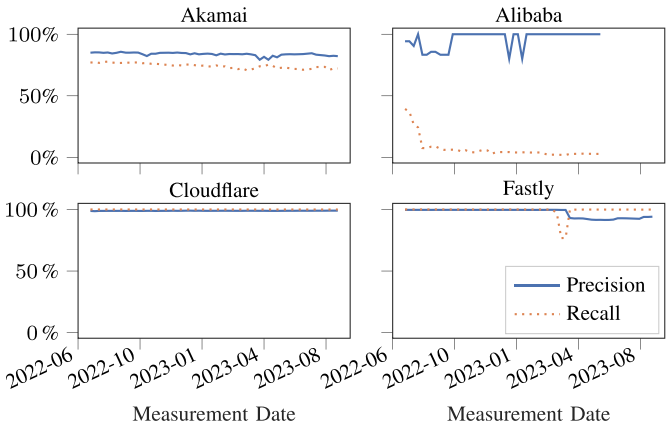


Fig. 4. Evaluation of a 40% threshold CDN server classifier. It was trained with TLS fingerprints from weeks $[1..n]$ and evaluated on week n.

CDNs turned out to be valid caches of the CDNs from Akamai and Cloudflare outside of their respective networks. We will discuss these targets in more detail in VI-D.

A multi-label classifier can be built using these data, according to Section III-C, by identifying a CDN server purely from its TLS fingerprints. To find a suitable classification threshold, we present an overview of the precision and recall for different thresholds in Fig. 3. Both metrics were evaluated according to the definition in Section III-C on our whole dataset to indicate a fitting threshold. Based on Fig. 3 we selected a threshold of 40% for the rest of this section. The threshold was a trade–off between a high detection precision and the ability to detect caches outside of a CDN AS (labeled as non-CDN server in the training set, falsely reducing the detection probability). We can see that a threshold of 100% was suitable for neither CDN, either because our initial training labels were imperfect (in case of a wrong mapping of IP addresses to ASs) or because the fingerprints were not entirely unique to each CDN.

Having a reasonable threshold allowed us to evaluate the multi-label classifier in more detail and over time. It was trained with the data from weeks $[1..n]$ and evaluated using the data from week $n + 1$. Servers in the training set were labeled CDN servers if their IP address was located within the respective CDN AS, returned a valid certificate, and had the appropriate HTTP Server header (according to Gigis et al. [24]). To incorporate the CDN servers outside of the CDN AS, we additionally labeled servers as CDN servers in the evaluation set if we could validate them as CDN cache with our TLS scanner. The predicted classes were always unambiguous because the fingerprints of the CDNs did not overlap. We evaluated precision and recall for each week individually. Fig. 4 illustrates the CDN classification results. The precision was high, with an average of 84% for Akamai, 99% for Cloudflare, and 97% for Fastly. The latter two had networks that were much more uniform and easily clusterable than those of Akamai. However, in contrast to our results from 2022 [10], the caches we detected from the Alibaba CDN reduced over time, and after June 5, 2023, we could no longer detect any targets. We observed a rising amount of servers responding with fingerprints we could attribute in most other cases to Tengine or nginx webservers (inferred from the HTTP header). These Web servers are not only deployed by Alibaba; thus, they cannot infer a CDN cache. It is possible that we observed a gradual technology shift in the CDN infrastructure. Additionally, Alibaba and Akamai have a more diverse cloud portfolio than the other examined companies, which could be the reason for the lower precision and recall.

We observed an interesting drop in the recall for Fastly in March 2023. A Server Behavior change caused the drop because we suddenly observed new behaviors not covered by the fingerprints from the previous measurements. Changing Server Behaviors could relate to a software update or other change in their infrastructure. We saw the new behaviors in the later measurements stabilizing the recall. Therefore, a potential fingerprint database must be regularly updated for the best performance.

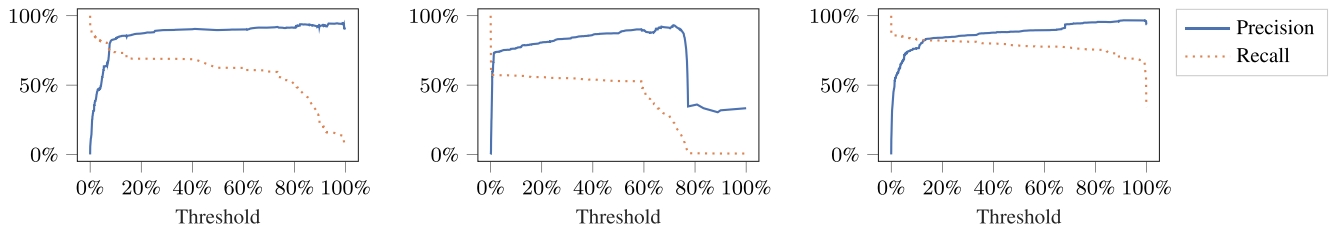
Sometimes, our fingerprints detected minor differences among the deployments of the same CDN. In these cases, the approach was too specific for the general use case to detect just the CDN. We mitigated the problem by mapping multiple fingerprints to each CDN, covering all of these variations. To detect Akamai, Alibaba, Cloudflare, and Fastly, we have used 397, 2, 2733, and 1513 fingerprints, respectively.

In summary, with active TLS fingerprinting, large CDN deployments can be identified because they share a mutual TLS behavior. The precision was above 97% for some CDNs, and we have found several CDN caches in unexpected ASs. After showing that the approach works with major known deployments, we will apply it to a much smaller sample size, identifying potentially malicious C2 servers.

D. Case Study: Identifying C2 Servers

Aside from identifying CDN deployments, TLS fingerprinting can identify and track potentially malicious targets like C2 servers.

We used blocklists containing C2 servers as an indicator of malicious behavior. Table V presents the measurement results for all servers from the blocklists grouped by C2 label (listing only successfully fingerprinted targets). The



(a) TLS Fingerprint (10 CHs) (b) HTTP Data (Server Header) (c) Combined (10 CHs + HTTP Data)

Fig. 5. Precision and recall to identify new observations on our input lists as C2 servers using the data described in Fig. 5abc as input for the classification.

TABLE V
FINGERPRINTING RESULTS FOR SUCCESSFULLY SCANNED C2 SERVERS. COMBINING OUR FINGERPRINT (FP) WITH HTTP DATA RESULTS IN MORE FPS AND DISTINCT TARGETS (TAR.) UNIQUE TO A C2 LABEL

C2 Label	Total		Unique[1]		Unique (+HTTP)[2]		New Obs.[3]	
	Tar.	FPs	Tar.	FPs	Tar.	FPs	Total	Known FP
QakBot	604	23	3	3	37	4	562	549
BumbleBee	246	38	0	0	3	3	189	176
Emotet	188	27	1	1	2	2	75	64
Ransomware[4]	151	20	2	2	44	36	123	115
Dridex	57	5	0	0	0	0	8	6
Other	44	27	5	5	12	10	38	11
AsyncRAT[4]	36	4	0	0	0	0	35	32
Gootkit[4]	28	4	0	0	1	1	22	21
DCRat[4]	16	4	0	0	0	0	16	12
Pikabot	16	1	16	1	16	2	16	2
Other	44	27	31	18	33	23	38	11

[1] Subset of targets with a FP unique to a C2 label.
[2] Same as "Unique", only adding the HTTP Server header to the FP.
[3] Subsets of C2 targets that were added to the blocklists during our measurement study. "Known FP" are the new observations with a FP already seen in combination with the respective C2 label.
[4] Labels from the SSLBL [42], the rest was from the Feodo Tracker [43].

Ransomware, AsyncRAT, Gootkit, and DCRat labels are from the SSLBL [42]. The remaining labels are from the Feodo Tracker [43]. Several fingerprints are unique to a specific type of C2 server. However, these unique fingerprints cover only a tiny part of all observations. Unique fingerprints gradually increase by combining them with additional HTTP data (i.e., the HTTP Server header containing values like `nginx` or `Apache/2.4.18`). New servers added to the blocklists repeatedly had the same fingerprint as past servers; i.e., 9% of the targets added during the week $n+1$ had fingerprints already observed for this label during weeks $[1..n]$. In other words, fingerprinting could identify those servers.

This work uses the threshold-based classifier from Section III-C to decide whether or not a server is a C2 server from a blocklist. A server was labeled a "C2 server" if its IP address was on the Feodo Tracker or the certificate was on the SSLBL. We evaluated the classifier on every new target added to the top list or blocklist during week $n+1$ based on the training data from weeks $[1..n]$. We evaluated the precision and recall of the classifier for each threshold according to Section III-C. The threshold serves as a tuning parameter; e.g., selecting a value of 80% means a fingerprint must be observed more than 8% from C2 servers such that a new observation with this fingerprint is classified as a C2 server.

Fig. 5 shows the results for three data sources used as input for the classification. The classifier performance significantly increases if we add HTTP data to the fingerprints by concatenating the TLS fingerprints with the HTTP Server header and using the result as a fingerprint. The HTTP Server header alone is not ideal for a classifier, but when combined, it achieves a maximum precision of 9% for 7% of the added C2 servers and a threshold of 91%. Interestingly, the highest precision was not achieved with a threshold of 10% because several good-performing fingerprints were only partially unique to blocklist servers. The lower recall indicates that our fingerprinting was not fine-grained enough to detect the differences in the deployments needed to identify all C2 servers. Augmenting the fingerprints with HTTP data was our solution to improve the granularity for more effective fingerprinting.

This analysis demonstrates that many C2 servers had unique TLS behaviors that allowed us to identify them. We presented how a classification of these servers is possible on a large scale and that such an approach can achieve high precision. Furthermore, a potential fingerprint database for C2 servers would live much longer than IP addresses on a blocklist, which means they can provide valuable information about newly deployed C2 servers until their IP address is publicly known.

E. Multi-Label C2 Detection

The previous section demonstrated the general feasibility of C2 detection using active TLS fingerprinting. Moreover, the unique C2 fingerprints from Table V indicate that it should be possible to predict the type of C2 server. A more fine-grained classifier can help explain the decision of the C2 classifier and provide additional valuable information.

We performed the same classification from Section V-D for each C2 label (every new observation is classified only based on the data from previous weeks). Only this time, we calculated the evaluation metrics on the combined data from every week. This analysis focuses only on four selected thresholds: 50%, 70%, 90%, and 100%. Table VI presents the precision and recall per threshold and label. We can see that the high precision from Table V was due to the detection of a few C2 server types, especially QakBot and BumbleBee. We could not classify some labels (i.e., GootKit and PikaBot) and others above a certain threshold (i.e., Emotet and Dridex). In these cases, the fingerprints obtained from the new observations were either unknown to us or seen too often (depending on the threshold) from servers on top lists. Note that the sample size of the evaluation set was low for some labels;

TABLE VI
PRECISION (P) AND RECALL (R) FOR A THRESHOLD (TH.)-BASED CLASSIFICATION PER C2 LABEL ON COMBINED DATA (FINGERPRINT + HTTP). SOME COMBINATIONS RESULTED IN NO PREDICTION (EMPTY)

C2 Label	50% Th.		70% Th.		90% Th.		100% Th.	
	P[%]	R[%]	P[%]	R[%]	P[%]	R[%]	P[%]	R[%]
QakBot	99	97	99	97	99	97	99	47
BumbleBee	96	81	99	81	99	79	100	31
Emotet	84	85	88	85				
Ransomware	46	28	50	18	50	14	46	12
Dridex	6	25						
AsyncRAT	71	63	61	31	67	23	89	23
Gootkit								
DCRat	29	12	33	6	50	6	50	6
Pikabot								

therefore, the calculated precision should be treated with care. For example, we found only 35 new observations labeled as AsyncRAT (as listed in Table V), and the recall of 23% means we successfully identified 8 of them. Arguably, more than such a small sample size is needed to generalize the precision of 89% for the 100% threshold.

This section demonstrated that active TLS fingerprinting can be used beyond a simple binary classification and can distinguish the type of C2 servers. We showed that detecting some C2 server types works well, with a precision above 99%, while other labels could not be detected at all.

F. Comparison With JARM and DissecTLS

After analyzing the applicability of active TLS fingerprinting for different use cases, the subsequent paragraphs look at the performance of JARM [8] and DissecTLS [20], which also allow the fingerprinting of TLS servers. While JARM uses similar data to our approach, we show that the data extracted from the TLS handshakes and the CH selection of this work provide an improved base for fingerprinting. DissecTLS surpasses both tools at the cost of its increased probing, as explained in Section II-A.

We scanned every target with the CHs used by JARM, the empirically optimized ones from this work, and the DissecTLS scanner. Thus, we can compare the three approaches. We did not use the open-source JARM script directly because it could not scan the number of targets on our hardware fast enough. Instead, we have used our scanner with the JARM CHs and extracted the subset of features that JARM uses to construct its fingerprints from our data. In particular, we stripped fingerprints from alerts, any TLS message besides the Server Hello, and any extension data besides the ALPN (i.e., the IDs and the order of the extensions remained intact, and we only removed the data contained in these extensions). Table VII compares how selecting features and CHs affects the fingerprinting results. DissecTLS works fundamentally differently; thus, we can only evaluate the overall performance instead of the two dimensions. The improvements proposed herein consistently provide better results while maintaining the number of requests necessary for fingerprinting the same (i.e., 10 CHs). In total, this work can differentiate 62% more Server Behaviors than JARM. Considering the C2 servers, the improved differentiation resulted in 15 unique C2 behaviors and 7.8 times more C2 servers identifiable with these unique behaviors. In contrast, DissecTLS identified 2.4 times more Server Behaviors than EFACTLS, resulting in 23 more uniquely identifiable C2 behaviors. "Unique" means we observed no overlap with any server found on a top list. Interestingly, the number of distinct fingerprints suggests our feature selection had a higher impact on the improved detection than the used CHs. In contrast, the Entropy and number of unique C2 fingerprints indicate the opposite was the case.

TABLE VII
COMPARING THE EFFECTIVENESS OF FINGERPRINTS (FPS) OBTAINED WITH EFACTLS, JARM, AND DISSECTLS CONSIDERING BOTH THE DIMENSIONS OF FEATURE SELECTION AND USED CHS

Feature selection	JARM		EFACTLS		DissecTLS
CH generation	JARM	EFACTLS	JARM	EFACTLS	
Unique FPs	41.8k	50.2k	57.0k	67.5k	159.7k
Entropy	5.9	7.0	6.0	7.2	8.0
Unique C2 FPs	8	16	14	23	52
Unique C2 targets	32	242	38	249	306

In conclusion, TLS fingerprinting tools like JARM can benefit from the advanced feature extraction and the systematic design of the CHs proposed in this work to improve the approach's effectiveness. Dynamic scanning tools like DissecTLS can surpass both approaches; however, the increased effectiveness comes with higher resource usage. A more thorough comparison can be found in [20].

VI. DISCUSSION

With our TLS fingerprinting approach, we gained new insights into the Internet and found interesting relations among TLS servers. We discuss some of them in the following paragraphs.

A. Advanced Similarity Comparison

This work explicitly does not obfuscate any information as done by [6] and [8]. Keeping the syntactic information of each part of the fingerprint intact supports explainability, allows us to relate similar behaviors in the future, and adapt the fingerprints afterward (e.g., removing the Status Request extensions). Similar fingerprints can indicate deployments from an actor who has done minor configuration changes.

B. The Success of Random CHs

Initially, we used the standard CHs from the Go library and CHs mimicking popular browsers for fingerprinting. However, they could not extract enough information from servers to be effective in use cases because the requests were similar, focusing on a few popular TLS parameters. In contrast, the Random CHs were empirically optimized to distinguish servers and have unusual combinations and order of parameters. They vary in the combination of TLS versions, ciphers, ALPN values, and supported groups and, sometimes, are not realistic (e.g., offer ALPNs unsuitable for

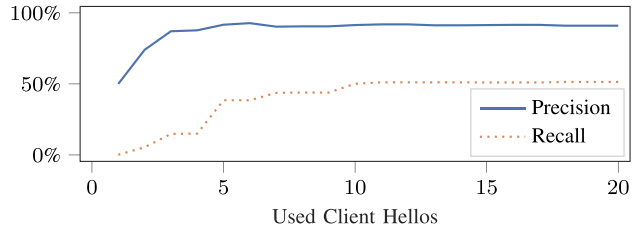


Fig. 6. Influence of the CHs on the C2 detection with an 80% threshold.

Web servers). Interestingly, two CHs use TLS 1.3, and none use TLS 1.2 as the `legacy_version`; neither conforms to the RFC [28]. In contrast, JARM uses more realistic CHs with fewer parameter variations. They defined them through systematic subsets (e.g., the top half) or a reversed order from a fixed input. In conclusion, the Random CHs are very successful for fingerprinting because they have a fuzzy character triggering more distinctive responses.

C. Adequate Number of CHs for C2 Detection

We designed the 10 CHs as a general-purpose configuration to provide a good base for classification. However, for specific use cases, they can be different. We scanned every server with 10 optimized and 10 JARM CHs; hence, we can recompute the classification performance shown in Fig. 5a using up to 20 CHs as input for an 80% threshold. We used a similar optimization strategy to find the next CH for the classification input, as in Section IV. We chose the Entropy as a maximization metric because Table VII revealed that the Entropy correlates better with the effectiveness of the C2 detection than the number of unique fingerprints. While the precision was high after three CHs, scanning with additional CHs mainly increased the number of classified servers (visible in the increasing recall). 18 CHs achieved the maximum precision and recall, but the gain was minimal after 10 CHs. Interestingly, two of the first 10 CHs were from JARM, providing a slightly higher precision (+0.8%) than our 10 general-purpose CHs To conclude, for the C2 detection use case, multiple CHs are necessary, but a few less than 10 would have also provided good results. Additionally, future work could implement an adaptive scanning approach where additional requests are only sent to a server if the precision of its current classification needs to be higher or the Entropy of server's fingerprint is too low.

D. CDN Caches in a Foreign AS

We could not correctly identify all the reasons why some domains resolved to CDN caches in a foreign AS. In particular, 56% were located in an AS from Episerver, Amazon, China Unicom, China Mobile, or Render. Interestingly, the whole Render AS was proxied through Cloudflare, and every IP address in the AS functioned as a CDN cache. Cloudflare was the only upstream provider for the Render AS (checked with [47]), effectively redirecting all traffic to the Render AS to Cloudflare via the BGP. However, not every other AS that seemed to contain a Cloudflare cache was redirected via BGP. We may have observed the effect of operators trying to remain in control of their traffic flow (e.g., by deploying a Meta-CDN [48]) because 93% domains pointing to a CDN cache in a foreign AS used a nameserver unrelated to the CDN. To our knowledge, Cloudflare does not operate CDN caches outside of their ASs. At least 11 of these caches were reverse proxies set up by third parties; investigated with a tracing endpoint[3] suggested to us by Cloudflare. Additionally, our data contained 204 outlier domain names resolving to the public Domain Name System (DNS) resolver `1.1.1.1`, which was also a cache to the Cloudflare network. 59% of these domain names used a Cloudflare nameserver. In contrast to Cloudflare, Akamai has deployed CDN caches in more than 1k foreign ASs to localize their traffic [49]. However, we detected just 16 ASs because we did not scan the full IPv4 address space, but IP addresses resolved from the top list. Akamai uses the DNS to distribute the load on their servers [49]. We assume, we saw these ASs because our scan traffic was not always directed to the closest CDN cache but distributed across servers in multiple ASs.

E. CDN Inconsistencies

Some CDN caches were inconsistent in their responses because not every IP address successfully responded to every requested domain name. In the end, multiple domain names were necessary to validate all caches. For Cloudflare, it was only a single IP address located in China. For Akamai, bigger clusters were visible and multiple domain names were needed to verify them.

F. Unstable Fingerprints

Some targets had inconsistent fingerprints that could be caused by the server or more complex setups. Sometimes, we saw indicators of load balancers in the HTTP Server header, indicating that the actual fingerprinted server changed during the scan process. An inconsistent server is also the main limitation of our approach because it relies on multiple TLS connections to connect to the same Server Behavior. However, inconsistent servers were rarely an issue (Section V-B).

G. Limitations of the Information Metrics

In this work, we used the number of fingerprints and the Entropy as metrics to compare the effectiveness of fingerprinting approaches. However, both are only useful metrics if the obtained fingerprints represent real-world server characteristics and remain stable for the same Server Behavior. Unstable fingerprints could trick both metrics into an unjustified high value. Unstable fingerprints can happen either because the methodology has flaws (e.g., it contains session-specific information and produces a new fingerprint in each connection) or because the servers produce a high entropy (e.g., by shuffling the order of TLS extensions or other parameters). Entropy can only compare the effectiveness of fingerprinting approaches if each approach is guaranteed to

[3]https://cloudflare.com/cdn-cgi/trace

provide only useful fingerprints. In EFACTLS, we confirm the usefulness of our fingerprints with our measurement studies. JARM uses a subset of our data, which means their fingerprints are also useful. The DissecTLS [20] paper also shows its applicability for fingerprinting with a measurement study. Consequently, the proposed metrics allow a comparison of the approaches in Table VII; however, additional tools must be compared with care.

H. Alternative Comparison Metrics

We selected the Entropy to model the amount of information obtained via active fingerprinting approaches. Initially, we used the number of fingerprints as metric; however, Section V-F indicates that Entropy is a better metric because it correlates more with the number of detected C2 servers. Although alternatives to Entropy were proposed it remains "the main tool in the analysis of the concept of information" [50] since Shannon's publication in 1948 [29]. An extension to our use of Entropy would be to model Conditional Entropy that considers already learned information. For example, a fingerprint might provide only a little information if we already know from the IP address prefix that the Server Behavior is most likely from a Cloudflare CDN cache. However, Conditional Entropy drastically increases the complexity of the metric because of the many possibilities to model the condition. An alternative metric is to compare fingerprinting approaches based on the performance of use cases; e.g., the DissecTLS work [20] compares approaches based on precision and recall of a C2 detection. However, such classification metrics rely on the quality of a ground truth.

Entropy has the advantage of providing a simple and intuitive metric based only on the data source itself. Using Entropy, Ground truth is optional and the metric can be calculated before an actual use case is known.

VII. Conclusion

This work proposed a methodology for acquiring and leveraging TLS metadata through large-scale active measurements. Two measurement studies conducted over a year on the Tranco top list and two C2 blocklists support the value of this approach. The two studies evaluate the detection of CDN and C2 servers. The precision in classifying new C2 servers added to the blocklists reached 97% and 99% for some C2 families. Depending on the CDN and its infrastructure, the average detection precision ranged from 83% (Akamai) to more than 97% (Cloudflare and Fastly). Additionally, we identified 555 IP addresses to serve CDN content outside of the AS operated by the CDN.

The results were obtained with a reasoned selection of features extracted from TLS handshakes and using multiple scanning probes to construct fingerprints of the TLS stack on servers. These probes were empirically optimized to provide maximum information while minimizing measurement time and the impact on targets.

This paper describes in detail how effective active TLS fingerprinting can be conducted and demonstrates the applicability of the approach to real-world classification problems, such as C2 detection, demonstrating its relevance in security contexts. Moreover, the extended feature extraction and improved CH design can improve existing active TLS fingerprinting tools while maintaining their scanning effort. Given that the approach is independent of the actual CHs, future research may explore tuning CHs for specific use cases or dynamically adapt them per server.

References

[1] M. S. Jelle van Haaster, R. Gevers, and M. Sprengers, *Cyber Guerilla*. Cambridge, MA, USA: Elsevier, 2016.

[2] C. Labovitz, "Internet traffic 2009-2019," in *Proc. Asia Pac. Reg. Internet Conf. Oper. Technol.*, 2019, pp. 1–26.

[3] P. Kotzias, A. Razaghpanah, J. Amann, K. G. Paterson, N. Vallina-Rodriguez, and J. Caballero, "Coming of age: A longitudinal study of TLS deployment," in *Proc. ACM Int. Meas. Conf. (IMC)*, 2018, pp. 415–428.

[4] B. Anderson and D. A. McGrew, "Accurate TLS fingerprinting using destination context and knowledge bases," 2020, *arXiv:2009.01939*.

[5] M. Husák, M. Cermák, T. Jirsík, and P. Celeda, "Network-based HTTPS client identification using SSL/TLS fingerprinting," in *Proc. 10th Int. Conf. Availabil., Rel. Security*, 2015, pp. 389–396.

[6] J. Althouse, J. Atkinson, and J. Atkins (Salesforce, San Francisco, CA, USA). *TLS Fingerprinting With JA3 and JA3S*. 2019. [Online]. Available: https://engineering.salesforce.com/tls-fingerprinting-with-ja3-and-ja3s-247362855967

[7] S. Gallagher (Sophos, Abingdon, U.K.). *Nearly Half of Malware Now Use TLS to Conceal Communications*, 2021. [Online]. Available: https://news.sophos.com/en-us/2021/04/21/nearly-half-of-malware-now-use-tls-to-conceal-communications/

[8] J. Althouse, A. Smart, R. Nunnally Jr., and M. Brady (Salesforce, San Francisco, CA, USA). *Easily Identify Malicious Servers on the Internet With JARM*. 2020. [Online]. Available: https://engineering.salesforce.com/easily-identify-malicious-servers-on-the-internet-with-jarm-e095edac525a

[9] "Data definitions." Censys.io. Accessed: Oct. 6, 2023. [Online]. Available: https://search.censys.io/search/definitions

[10] M. Sosnowski et al., "Active TLS stack fingerprinting: Characterizing TLS server deployments at scale," in *Proc. Netw. Traffic Meas. Anal. Conf. (TMA)*, 2022, pp. 1–9.

[11] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, "Tranco: A research-oriented top sites ranking hardened against manipulation," in *Proc. Netw. Distrib. System Security Symp. (NDSS)*, 2019, pp. 1–15.

[12] "We will be retiring Alexa.com on may 1, 2022." Alexa Internet, Inc. 2021. [Online]. Available: https://web.archive.org/web/20211208204723/https://support.alexa.com/hc/en-us/articles/4410503838999

[13] L. G. Greenwald and T. J. Thomas, "Toward undetected operating system fingerprinting," in *Proc. USENIX Workshop Offens. Technol.*, 2007, pp. 1–10.

[14] Z. Shamsi, A. Nandwani, D. Leonard, and D. Loguinov, "Hershel: Single-packet OS fingerprinting," *IEEE/ACM Trans. Netw.*, vol. 24, no. 4, pp. 2196–2209, Aug. 2016.

[15] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, Seattle, WA, USA, 2009.

[16] J. Zirngibl, F. Gebauer, P. Sattler, M. Sosnowski, and G. Carle, "QUIC library hunter: Identifying server libraries across the Internet," 2023, *arXiv:2308.15841*.

[17] J. Iyengar and M. Thomson, "QUIC: A UDP-based multiplexed and secure transport," Internet Eng. Task Force, RFC 9000, May 2021.

[18] A. Diquet. "SSLyze." Accessed: Oct. 6, 2023. [Online]. Available: https://github.com/nabla-c0d3/sslyze

[19] "Testing TLS/SSL encryption, Dirk Wetter." Accessed: Oct. 6, 2023. [Online]. Available: https://testssl.sh

[20] M. Sosnowski, J. Zirngibl, P. Sattler, and G. Carle, "DissecTLS: A scalable active scanner for TLS server configurations, capabilities, and TLS fingerprinting," in *Proc. Passive Act. Meas. (PAM)*, 2023, pp. 110–126.

[21] R. Moore. "TLS prober-an SSL/TLS server fingerprinting tool." Accessed: Oct. 6, 2023. [Online]. Available: https://github.com/WestpointLtd/tls_prober

[22] T. Chung et al., "Is the Web ready for OCSP must-staple?" in *Proc. ACM Int. Meas. Conf. (IMC)*, 2018, pp. 105–118.

[23] D. E. Eastlake, "Transport layer security (TLS) extensions: Extension definitions," Internet Eng. Task Force, RFC 6066, 2011.

[24] P. Gigis et al., "Seven years in the life of Hypergiants' off-nets," in *Proc. ACM SIGCOMM*, 2021, pp. 516–533.

[25] E. Papadogiannaki and S. Ioannidis, "Pump up the JARM: Studying the evolution of Botnets using active TLS fingerprinting," in *Proc. Int. Symp. Comput. Commun. (ISCC)*, 2023, pp. 764–770.

[26] R. Holz et al., "Tracking the deployment of TLS 1.3 on the Web: A story of experimentation and Centralization," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 50, no. 3, pp. 3–15, 2020.

[27] (IANA, Los Angeles, CA, USA). *Transport Layer Security (TLS) Extensions*. Accessed: Sep. 12, 2023. [Online]. Available: https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml

[28] E. Rescorla, "The transport layer security (TLS) protocol version 1.3," Internet Eng. Task Force, RFC 8446, 2018.

[29] C. E. Shannon, "A mathematical theory of communication," *Bell System Tech. J.*, vol. 27, pp. 379–423, Oct. 1948.

[30] N. Etemadi, "An elementary proof of the strong law of large numbers," *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, vol. 55, pp. 119–122, Feb. 1981.

[31] Q. S. Birk Blechschmidt. "MassDNS." 2023. Accessed: Jan. 19, 2024. [Online]. Available: https://github.com/blechschmidt/massdns

[32] "Unbound." NLnet Labs. 2024. Accessed: Jan. 19, 2024. [Online]. Available: https://www.nlnetlabs.nl/projects/unbound

[33] O. Gasser, M. Sosnowski, P. Sattler, and J. Zirngibl. "TUM goscanner." Accessed: Jan. 19, 2024. [Online]. Available: https://github.com/tumi8/goscanner

[34] J. Amann, O. Gasser, Q. Scheitle, L. Brent, G. Carle, and R. Holz, "Mission accomplished? HTTPS security after Diginotar," in *Proc. ACM Int. Meas. Conf. (IMC)*, 2017, pp. 325–340.

[35] Z. Durumeric, E. Wustrow, and J. A. Halderman, "ZMap: Fast Internet-wide scanning and its security applications," in *Proc. USENIX Security Symp.*, 2013, pp. 605–620.

[36] D. Dittrich and E. Kenneally, *The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research*, U.S. Dept. Homeland Secur., Washington, DC, USA, 2012.

[37] C. Partridge and M. Allman, "Addressing ethical considerations in network measurement papers," *Commun. ACM*, vol. 59, no. 10, pp. 58–64, 2016.

[38] "Active TLS fingerprinting: Additional material." 2024. [Online]. Available: https://tumi8.github.io/active-tls-fingerprinting/

[39] (IANA, Los Angeles, CA, USA). *Transport Layer Security (TLS) Parameters*. Accessed: Oct. 6, 2023. [Online]. Available: https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml

[40] D. R. L. Brown (SECG, Chicago IL, USA). *SEC 2: Recommended Elliptic Curve Domain Parameters*. 2010. [Online]. Available: http://www.secg.org/sec2-v2.pdf

[41] M. Sosnowski et al., 2024, "EFACTLS measurement data," Dataset, [Online]. Available: https://doi.org/10.14459/2024mp1733820

[42] (abuse.ch, Zurich, Switzerland). *SSL Certificate Blacklist*, Accessed: Oct. 6, 2023. [Online]. Available: https://sslbl.abuse.ch/

[43] (abuse.ch, Zurich, Switzerland). *Feodo tracker*. Accessed: Oct. 6, 2023. [Online]. Available: https://feodotracker.abuse.ch/

[44] N. Sullivan (Cloudflare, San Francisco, CA, USA). *High-Reliability OCSP Stapling and Why It Matters*. 2017. [Online]. Available: https://blog.cloudflare.com/high-reliability-ocsp-stapling/

[45] *Route Views Archive Project*. Univ. Oregon, Eugene OR, USA, Accessed: Jan. 18, 2024. [Online]. Available: https://routeviews.org/

[46] H. Asghari, "Pyasn." Accessed: Jan. 18, 2024. [Online]. Available: https://github.com/hadiasghari/pyasn

[47] (Hurricane Electr., Fremont, CA, USA). *Internet Services*. Accessed: Oct. 6, 2023. [Online]. Available: https://bgp.he.net/

[48] O. Hohlfeld, J. Rüth, K. Wolsing, and T. Zimmermann, "Characterizing a Meta-CDN," in *Proc. Passive Act. Meas. (PAM)*, 2018, pp. 114–128.

[49] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai network: A platform for high-performance Internet applications," *ACM SIGOPS Oper. Syst. Rev.*, vol. 44, no. 3, pp. 2–19, 2010.

[50] M. J. Schroeder, "An alternative to entropy in the measurement of information," *Entropy*, vol. 6, no. 5, pp. 388–412, 2004.

Markus Sosnowski received the degree in computer science from the Technical University of Munich, Germany, and the M.Sc. degree in the Elite Master Programme Software Engineering, which is offered within the framework of the Elite Network of Bavaria by the University of Augsburg, the Ludwig-Maximilians-Universität München, and the Technical University of Munich, Germany. He is currently pursuing the Ph.D. degree in computer science with the Technical University of Munich, where he is also currently a Research Associate. His research interests lie in network modeling, software engineering, the TLS ecosystem, and Internet measurements. He currently focuses on security-related insights derived from active scanning and modeling of the TLS ecosystem.

Johannes Zirngibl received the Master of Science degree in computer science from the Technical University of Munich in 2019, including a semester abroad at the University of Illinois Urbana–Champaign. He is also co-leading the Global INternet Observatory, Chair of Network Architectures and Services, Technical University of Munich which continuously operates various ongoing measurement studies. His Ph.D. thesis focuses on the deployment and analysis of Internet-wide measurements to provide insights into network and protocol deployments. His analyses focus on the deployment of QUIC, biases in results, such as the effect of domain parking, and the impact of aliased prefixes and DNS injections on IPv6 scans. His research covers multiple protocols of the network stack, including QUIC, IPv6, and DNS.

Patrick Sattler received the Master of Science degree in computer science from the Technical University of Munich (TUM) in 2019. During his study, he went to Dublin to do an internship at the Network Monitoring Group by AWS. In 2019, he started his position as a Research Assistant with the Chair of Network Architectures and Services led by Prof. G. Carle. As part of his work, he is also co-leading the Global INternet Observatory Research Group, TUM, which operates several long-term measurement studies. His focus thereby lies on the deployment of services and the DNS architecture. His research focuses on Internet-wide measurement studies to observe the infrastructure and deployment of networks.

Georg Carle received the degree in electrical engineering from the University of Stuttgart, with studies abroad at Brunel University, London, and ENST Paris (currently, Telecom ParisTech), and the Ph.D. degree in computer science from the University of Karlsruhe (currently, Karlsruhe Institute of Technology) in 1996. He holds the Chair of Network Architectures and Services, Technical University of Munich (TUM), Germany. He conducts research on high-performance network technologies, Internet measurements, and network security. Subsequently, he was a Postdoctoral Scientist with Institut EURECOM, Sophia Antipolis, France, and the Fraunhofer Institute for Open Communication Systems FOKUS, Berlin. In 2003, he joined the University of Tübingen as a Full Professor, and in 2008, he accepted an offer from TUM.

Claas Grohnfeldt received the degree in industrial mathematics from the University of Bremen, and the Ph.D. degree from the Technical University of Munich, Germany. He joined the Munich Research Center of Huawei Technologies in 2018, where he is currently a Principal Research Engineer addressing cybersecurity problems using mainly machine learning.

Michele Russo received the degree in electronics and telecommunications engineering from the University of Trento, Italy, and the master's degree in computer science from the Polytechnic University of Turin and Telecom ParisTech. He is currently a Research Engineer with Huawei Munich Research Center. His main research interests are the security of computer networks and the application of machine learning to security problems.

Daniele Sgandurra received the Ph.D. degree in computer science from the University of Pisa, with a dissertation focused on enforcing properties of programs through introspection. His professional experience spans academia (University of Pisa, Imperial College London, and Royal Holloway University of London), industry (Finmatica, and Huawei), and research centres (IBM Research and the Italian National Research Council). His research is focused on practical aspects of systems and software cybersecurity, analyzing potential threats across various architectural levels to propose effective countermeasures. His recent work delves into the ongoing arms race between malware and anti-malware, as well as the integration of AI in cybersecurity.